Modeling the Europa Pathfinder Avionics System with a Model Based Avionics Architecture Tool[1]

Marcus Traylor, Ronald Hall, Savio Chau
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr., Pasadena, CA 91109, USA

Adrian Whitfield,
I-Logix Inc.
Three Riverside Drive, Andover, MA 01810, USA

Abstract—In order to shorten the avionics architecture development time, the Jet Propulsion Laboratory has developed a model-based architecture simulatin tool called the the Avionics System Architecture Tool (ASAT). ASAT is consisted of a library of components models that can be quickly assembled into an avionics system model. The simulation of the avionic system is driven by a scenario script that captures the high level description of the spacecraft operations. Hence, the avionic system architecture can be validated and system level design problems can be uncovered before any hardware and software development. In addition, since the component models can easily be added, changed, or removed from the system model, this tool can also perform trade studies among architecture options. This tool has recently been applied to model the avionics architectures of the Europa Pathfinder mission study project. The results of that application are also reported in this paper.

TABLE OF CONTENTS

1. INTRODUCTION

Since space missions are expensive and risky, and space environments are very unforgiving, design mistakes are often extremely difficult or impossible to correct. As a result, mistakes in spacecraft system engineering can result not only in multi million dollar overruns, but also compromised science collection, mission redesign, and total loss of the spacecraft.

Over the years, the Jet Propulsion Laboratory (JPL) has developed a system engineering process for spacecraft development. This process includes mission conceptual development, system definition, and architecture design. While this process has been successfully applied to many flight missions, it has been costly and lengthy because it requires many meetings of system engineers in order to establish the requirements and specifications. Worse yet, these traditional textual requirements and specifications are often error prone, ambigious and sometimes contain many oversights and mistakes. With these specification problems often not realised until final test or even later, during the mission itself, the re-engineering effort involved can often be extremely expensive. Therefore, JPL has initiated a series of efforts to develop a set of system engineering tools to improve the process. One of these tools is the Avionics System Architecture Tool (ASAT).

ASAT focuses only on the avionics system of the spacecraft design. The process of avionics system design is similar to, and in fact driven by, the spacecraft design process. Hence, in the *mission concept phase*, the avionics system engineer has to estimate mass, power, volume, and cost of the avionics system based on the mission concept. The avionics system engineer may have to negotiate with the spacecraft system engineer if the estimates exceed the amounts allocated. Eventually, these estimates become the mission level requirements for the avionics system. In the *system definition phase,* system requirements are developed. These requirements are evaluated by the avionics system engineer, who then creates a high-level avionics architecture. Mass, power, volume, and cost estimates are revisited based on the high-level architecture. The system performance and reliability are estimated and compared against the mission requirements. In the *architecture design phase*, details of the avionics architecture are developed. Trade studies are also employed to consider different architecture options. Detailed requirements are generated for implementation. All estimates and functional correctness of the architecture are verified through design reviews [1]. Figure 1 outlines this process.

An experienced avionics system engineer can complete the mission concept and system definition phases in a relatively short time with the help of simple tools such as spreadsheets. This is because these two design phases can tolerate a rather large margin of error in the estimates and

[1] IEEE Aerospace Conference paper #4??

because many details of the spacecraft system have not been defined. This is because these two design phases can tolerate a rather large margin of error in the estimates and because many details of the spacecraft system have not been defined. In contrast, the architecture design phase involves generating details of the architecture where functional correctness is a major concern. The margin of error in the estimates also has to be tightened because it has direct impact on the cost and schedule in the implementation phase.

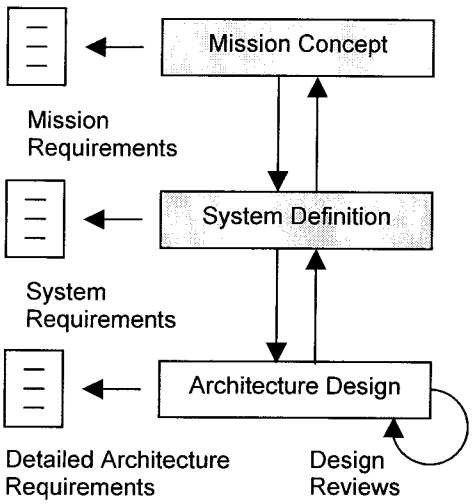

Figure 1 – Simplified Systems Engineering Process

The activities of the architecture design phase lend themselves to using executable architecture models as an analysis tool. Executable architecture models can verify architecture functions and estimate accuracy. They also allow the engineer to explore system dynamics such as interactions between components and time dependent behaviours. However, executable architectures and related development tools have their own requirements.

In order to support trade studies, the architecture model must be flexible and modular, made up of components that can be easily added, removed, or exchanged while verifying the consistency of the component interfaces. Such flexibility can also support reuse of architecture models. That is, once an architecture model is captured, the model itself or its components can be used again to model other avionics systems with sometimes only minimal changes. Furthermore, the simulation results of the architecture model should be easily captured and formatted to guide the development of the implementation requirements. Finally, since the architecture model is executable, it should be able to be integrated with other subsystem models to support

spacecraft level simulations. These simulations being used to verify command sequences in mission operations. Therefore, these requirements suggest a more sophisticated tool than a simple spreadsheet.

The role of ASAT in the spacecraft development process is depicted in Figure 2. The primary objective of ASAT is to help the avionics system engineers to develop architecture models to meet these needs.

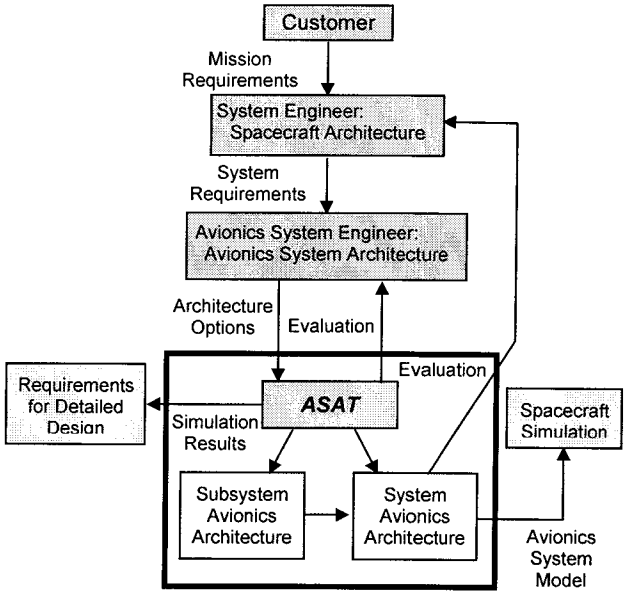

Figure 2 – Role of ASAT in Spacecraft Design Process

2. APPROACH

Initial ASAT development included the capture and refinement of users needs for a system engineering oriented tool suite. Interviews with members of the user community helped develop sets of needs in four main areas: ease of use, capability (vis-à-vis competing tools), validated components, and cost effectiveness. These needs were then analysed to determine sets of effectiveness measures in each area. These effectiveness measures are, in turn, evaluated to determine interrelationships, rank, and cost. The results being a set of prioritised requirements for ASAT. Table 1 describes how each of the users needs was sastisfied, with the most important appearing first in the table.

Based on feedback from the user group sessions discussed previously, it was clear that ASAT should focus initially on ease of use and cost effectiveness. To address these user concerns, the team developed two key mechanisms: plug-and-play and a standard data exchange infrastructure between core X2000 components. These mechanisms will be described in more detail later in this paper. To speed up the development of ASAT the team wanted to use an existing System Engineering modelling tool that could provide the necessary tool requirements addressed by the users. The next section of this paper describes the reasoning behind selecting Statemate Magnum from I-Logix to fulfil this need. Additional capabilities (see table 1), would then be added to the tool, to address the application specific requirements from the users.

In order to prove the concept is achievable, ASAT uses the X2000 avionics architecture as an example for prototype development. The X2000 is a technology program at JPL funded by NASA to develop system architecture and technologies for future spacecraft. The X2000 architecture is highly scalable and adaptable to a wide range of space missions. A typical instance of the X2000 architecture is shown in Figure 3. Therefore developing ASAT in conjunction with the X2000 will

demonstrate the capability to capture this architecture and then allow the user the ability to replace components or modify the architecture with minimum effort.

Table 1. The ASAT Approach to Meeting User Needs

User Need	Satisfaction/Effective Measure
Ease of Use	• Design and implement technologies common to all components that support user concepts of ease of use, including: • Plug-and-Play • Data Driven
Capability	• Design and implement architecture components of appropriate accuracy and fidelity • Design and implement support tools that allow users to do things competing tools do not support • Use development tools that support analysis of architecture dynamics
Validation	• Design and develop components based on standard specification (e.g. IEEE Std 1394a)
Cost Effectiveness	• Components are reusable • Architectures are reusable • Reduce architecture development time • Reduce trade study time

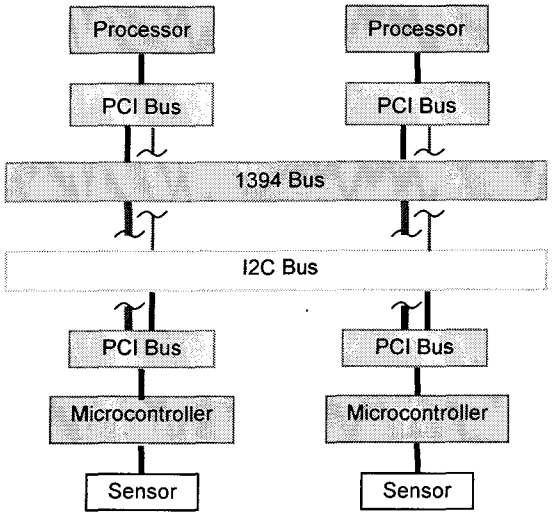

Figure 3 – X2000 Architecture Model

3. DEVELOPMENT OF THE TOOL SUITE

JPL has attempted model-based system engineering previously. Those models tried to describe the very detailed intefaces between all components used in the architecture model. Although this approach is not incorrect, subsequent experience has shown it is prohibitively time consuming as it does not lend itself to rapid design trades that are paramount in the *architectural design phase.*

Based on this observation, ASAT has taken a different approach. This new approach utilized a standard multi layer interface, communications protocol, infrastructure support functionality, architecture configuration file, and scenario files. The standard interface and protocol enable the component models to be connected in a plug-and-play fashion. This plug-and-play approach and standard interface allows system engineers to focus on the investigating system level issues and evaluating the avionics architecture design of the system rather than getting caught up on implementation specific issues. Thus, the system engineer can compose or modify architecture model for trade-off studies in quick turn around time.

The standard interface can be broken down into multiple layers as shown in Figure 4. The diagram shows the connections between three components. Each component features core, implementation layer and ASAT layers. The core layer represents the component's core functionality. For example, if a component represented an IEEE 1394 bus, the core would represent the dynamic behaviors, protocols, timing, etc that are associated with that bus. To describe the implementation layer and ASAT layer its best to consider an example architecture setup. In Figure 4, *component one* could represent a Flight computer, *component two* representing an IEEE 1394 bus and *component three* representing a Microcontroller. The implementation layers within a component describe the true interfaces, necessary protocol and timing functionality, within the implementation, that would be necessary to interface one component core to another component core. For example, if one was interfacing a Flight computer with a PCI bus, a host bridge interface would be necessary. Alternatively, interfacing the PCI bus with the IEEE 1394 bus would also require some dedicated hardware that could provide this capability. Finally, to facilitate a component plug-and-play architectural modelling environment with rapid design trades allowable, each component interfaces through the ASAT layer and the infrastructure support functionality. The ASAT layer describes a special communications protocol that allows all ASAT components within the architecture to communicate.

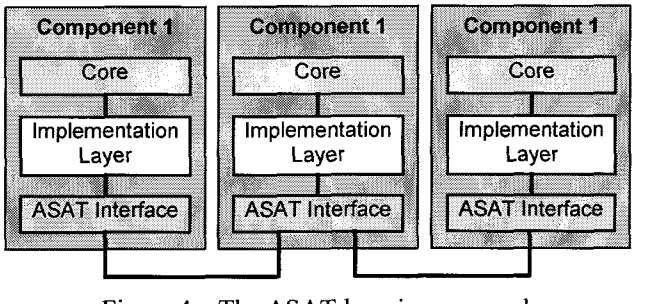

Figure 4 – The ASAT layering approach

The infrastructure provides the user with the development enviroment in which architecture models are developed and interfaced. The architecture configuration file defines how components are connected to each other. Any given architecture requires a correct configuration file. However, users can define multiple architectures using the same component set by defining different configuration files. For a typical simulation run, the components which

make up the system architecture will automatically find out how they are connected and interact with each other. This integrated architecture model will then be stimulated by a the scenario file and any measurements generated by the simulation will be captured for future analysis.

This approach reduces the process of building and testing an architecture into five steps:

1. Place the components in the architecture development environment
2. Connect each component to the architecture support infrastructure
3. Build an architecture configuration file describing component connectivity
4. Build scenario files to exercise the architecture
5. Run the scenario files through the architecture

This multiple layer approach can also reduce the development time of individual component models. When a new component model has to be developed and if the component is a variation of an existing component, then only the implementation layer need to be modified. If a similar component does not exist in the library, the core and implementation layers have to be developed but the interface layer will remain unchanged..

4. Results of the Prototype

The plug-and-play and common data exchange infrastructure has proven to be very effective. Tables 2 and 3 show component development time and X2000 architecture integration time respectively. Table 2 shows that the average time to build a component model was about 12 weeks. The conventional wisdom would expect the integration time would take at least that much time to integrate all the component models. However, Table 3 shows that the integration time for two variations of the X2000 architecture model, one has the I2C as the system bus (see the center bus in Figures 5) and the other with the PCI as the system bus. It is shown that the integration for the the PCI and I2C versions of the architecture model took only 1.5 days and 3 days, respectively. This experiment demonstrates the flexibiity of the plug-and-play infrastructure of ASAT.

Table 2. Component Model Development Time

	PCI	I2C	Processor	IEEE 1394a
Component Complexity	High	Medium	Medium	Very High
Intended Fidelity	High	High	Medium	Very High
Learn/ Specification Time	2-3 Weeks	1 Month	2 Weeks	3 Months
Build Time	1 Month	2 Months	3 Weeks	>1 Month
Level Of Modeling Expertise	Very Experienced	Beginner	Expert	Very Experienced

Table 3. X2000 Architecture Model Integration Time

	Communication Among Modeling Engineers	Integration	Test
PCI Architecture Example	0 [note]	0.5 Day	1 Day
I2C Architecture Example	0.5 Days	1 Days	1.5 Days

Note: The PCI model was used during the development of the plug-and-play infrastructure. Therefore, no learning of the PCI model is required in the architecture example.

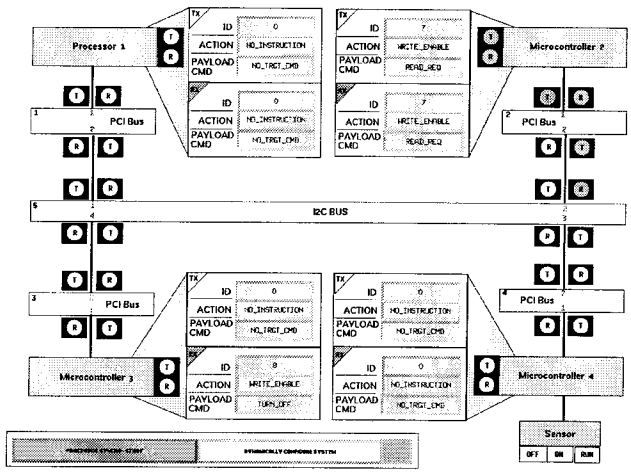

Figure 5 – ASAT Prototype Model

5. Modeling the Europa Pathfinder Avionics Architecture with ASAT

After the successful prototype, ASAT was used to model the Europa Pathfnder's avionics architecture. The Europa Pathfinder is a task to study the mission of landing on Europa, one of Jupiter's satellites that has a liquid ocean underneath the water ice crust and might be able to sustain life. The study includes developing a conceptual design for the avionics system of the lander.

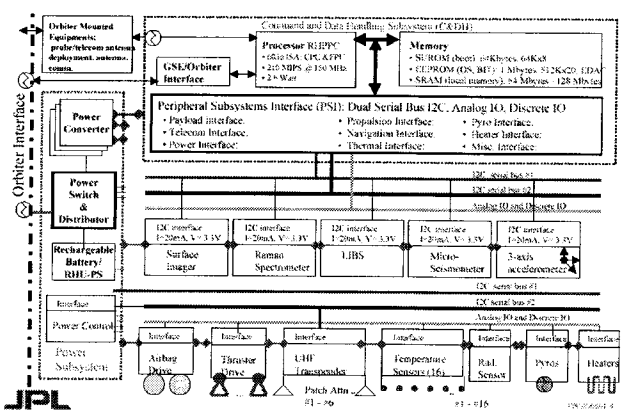

Figure 6 – Europa Pathfinder Avionics Architecture

The initial avionics system concept consists of a RAD6000 processor, subsystems such as telecommunication and power control and distribution, and a suite of instruments including a surface imager, a Raman spectrometer, micro-seismometer, radiation sensor, and

accelerometer. The Europa Pathfinder avionics system architecture is shown in Figure 6.

One important consideration for the Europa Pathfinder avionics system is the communication between the lander and the orbiter. The lander has a shape of a disc with a diameter of about 1.5 meters. The mission study has also chosen passive landing using airbags instead of active navigation. Since the surface of the Europa has many rifts, after the airbags are deflated and retracted, it is possible that the lander would fall into one of the rifts. In that case, the lander will have only a 8-minute time window to communicate with the orbiter. Consequently, much data including images have to be transmitted to the orbiter in this time window. This implies the interface between the processor and the telecommunication system has to have very high data rate. Another important consideration is the power consumption of the avionics system since the entire mission is powere by battery.

Based on these considerations, the mission study has proposed to use the high speed I2C bus as the interface between subsytems. The high speed I2C bus is a collisoin detection multi-master bus with a data rate of 3.4 Mbits/sec. There are two I2C buses in the architecture. The first I2C bus is dedicated to the interface between the processor and the telecom-munication subsystem. The second I2C bus is used as the system bus between the processor and all other subsystems.

Based on these inputs, the ASAT team has developed an architecture model for the Europa Pathfinder, which is shown in Figure 7. The architecture model is driven by a scenario script that captures the operations of the mission. The scenario provided by the Europa Pathfinder mission study team is shown in Figure 8. The scenario is translated into the ASAT scenario language. An example of the scenario script is shown in Figure 9.

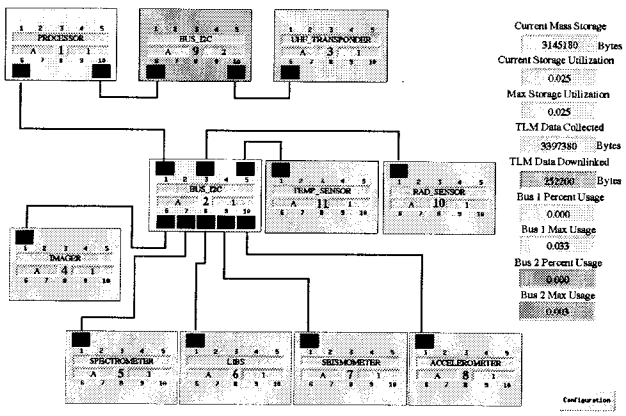

Figure 7 – ASAT Model for Europa Pathfinder Avionics

The effectiveness of the plug-and-play approach is proven once again. The architecture model was developed in 20 days, from the time when the ASAT team first received information to the completion of the working model. In fact, the majority of the time was spent on developing the high speed I2C bus model and the instrument models. The integration of the component models took only 2 to 3 days.

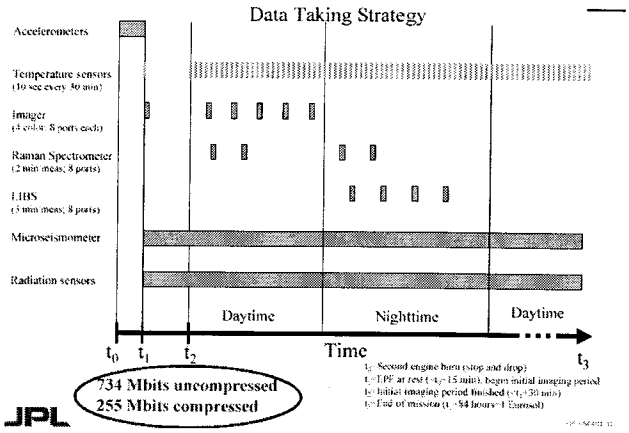

Figure 8 – Europa Pathfinder Data Collection Scenario

```
!
! Execute command at          0.000000
!
EXECUTION_TIME 0.0
MSG_SUBASSY_1 PROCESSOR
MSG_ID_1 1
PRIORITY_1 A
MSG_SUBASSY_2 ACCELEROMETER
MSG_ID_2 1
PRIORITY_2 A
PAYLOAD_CMD EUROPAPATHFINDER_SENDPERIODIC
ACTION WRITE_ENABLE
DIRECTION FORWARD
LENGTH 5
USER_TAG 1
MSG_ID 4
DATA_LENGTH    13.0
DATA_PERIOD    1.0
REPEAT      899
SENDTO_SUBASSY PROCESSOR
SENDTO_ID     1
SENDTO_PRIORITY A
END MSG
```

Figure 9 – ASAT Scenario Script for Europa Pathfinder

The simulation result has shown that the maximum utilization of the system high speed I2C bus was only 3%. That raised the question of whether a fast I2C bus (400 kbits/sec) can handle the bus traffic. Hence, the ASAT team replaced the high speed I2C model with a 400 kbits/sec model and run the simulation again. It was found that the bus utlization was only 17%. Since developing a 400 kbits/sec I2C bus is less expensive than a 3.4 Mbit I2C, this simulation suggested a less costly alternative to the original architecture.

However, in the 400 kbits/sec I2C bus architecture, we have also observed that the processor was not able to communicate with other subsystems when the imager is sending data to the processor. This was not observed in the high speed I2C bus architecture. After some analysis, we have found that this was caused by the differences in the arbitration scheme between these two versions of the I2C bus. In the 400 Mbits/sec I2C bus, the bus arbitration priority is based on the receving node address. Since the processor was assigned to the lowest address (highest priority), when the imager sent a large image to the processor, its data blocked all other nodes including the processor itself from sending their data onto the bus. On the other hand, since the high speed I2C bus arbitrates with the sending node address, the processor was able to preempt the imager. We solved this problem by changing

the scenario script, so that the image data are sent in small blocks to give the other subsystems opportunities to arbitrate for the bus.

Another consideration of using the 400 kbits/sec I2C bus is the bandwidth requirement between the processor and the telecommunication system. It is a fact that the 400 kbits/sec data rate would not be able to transmit all data to the telecommunication system in 8 minutes as required. On the other hand, we have noticed that only a few images would be collected in every 24 hours period. Hence, it is not too difficult to increase the buffer size in the telecommunication subsystem to store those images. This would allow data to be transmitted to the telecommunication subsystem at a lower rate but in a longer period of time.

Therefore, many observations and inspiratons were obtained from this architecture model, which was developed in a much short time than the traditional modeling approach at JPL. The architecture development time is expected to be significantly reduced when more component models would have been developed. In fact, since the 400 Mbits/sec I2C had been developed before this exercise, replacing the high speed I2C model with the 400 Mbits/sec I2C model took less than a day.

6. Conclusion

In this paper, we have described the process of system engineering in spacecraft avionics design and why a tool based approach that captures system dynamics is essential for providing robust design requirements. In order for us to provide a complete solution to our users we initially identified their concerns about the usefulness of a tool based solution and then responded by concentrating on key tool ease of use and cost effectiveness measures. In order for us to acomplish many of our users concerns we adopted a COTS systems engineering tool, Statemate Magnum, with a unique layered component standardised interface. To fully prove our concepts and ideas, the X2000 architecture was used as our technology demonstrator. From this we were able to demonstrate the efficiency of the approach and the results show that plug-and-play reduces architecture development and modification time. Then, we applied to tool to a real world avionics architecture and it was shown that the capture and demonstration of system dynamics can lead to many findings about an avionics architecture design.

7. Future Work

Although the plug-and-play infrastructure has significantly reduced the time to integrate components in the architecture model, the development of individual models still takes a relatively long time. Opportunities to reduce component development cycle time must be found and exploited. Ways to standardize protocols between components, such as data buffering and handshaking, to addreess component interoperability concerns must be investigated. It must also be stressed that the current product is only a proof of concept. A detailed analysis of the whole range of effectiveness measures in the users' areas of concern must be undertaken. This would lead to a robust set of requirements of an integrated system engineering tool such as ASAT.

One of the key concerns of our users is model verification and validation. A component validation/certification process must be developed to garner user trust. Verification is being addressed by systematically testing each and every behaviour of all components to verify our implementation. Once we are convinced that we have implemented the models properly (i.e. they do what we intended them to do), it is necessary to validate that our models yield accurate results. We are planning to do this through model design reviews, but also by comparing the results of the simulation against a hardware testbed currently being built at JPL. Our plan is to produce several scenarios that represent different architectures that can be simulated by our tool, and compare our results to identical scenarios executed by the hardware testbed. Each new component model added to our suite will be required to undergo similar verification and validation procedures. Therefore, a feature rich set of functional component models that have being fully verified and validated, will allow our users to perform accurate analaysis, yielding results that would be similar to using the real hardware architecuture.

Acknowledgement

This work has been funded by the Develop New Product program at the Jet Propulsion Laboratory, California Institute of Technology in association with I-Logix Inc. We would also like to acknowledge and thank the Europa Pathfinder Study Team, in particular Dr. Jacklyn Green, Dr. Wayne Zimmerman, and Dr. Wai-Chi Fang for their inputs to the Europa Pathfinder ASAT model.

Reference

[1] Tooraj Kia and Doug Bernard, Avionics Flight System Engineering, JPL, 1998

[2] Harel, David: Statecharts, A visual Formalism for complex systems in "Science of Computer Programming", 8 (1987) 231-274

Savio N. Chau *received his Ph.D. in computer science from the University of California, Los Angeles in 1989. He is a senior system engineer and the task manager of the X2000 Future Deliveries Project at the Jet Propulsion Laboratory. He is currently developing scalable multi-mission avionics system architectures and investigating techniques to apply low-cost commercial bus standards in highly reliable long-life spacecraft. His research areas include scalable distributed system architecture, fault tolerance system design, architecture modeling, and rapid integration of intellectual properties on ASICs.*

Ronald Hall *received his undergraduate degree in*

Computer Science and quantitative Economics at the University of California, San Diego, in 1984. He received his Master's degree in Business Administration at the University of Southern California in 1996. He has over 10 years experience in software engineering and software systems engineering in aerospace and healthcare environments. Currently, Mr. Hall is an avionics system engineer at the Jet Propulsion Laboratory where he works in the area of model based system engineering.

***Marcus Traylor** received a Master's degree in Electrical Engineering at the University of Texas at El Paso in 1993. He then came to the Jet Propulsion Laboratory where he spent five years working in modelling, simulation, and AI applications to simulation. For the last three years he has worked on statemate simulations of spacecraft avionics systems to improve the design process at JPL.*

***Adrian Whitfield** received an honours degree in Electrical and Electronic Engineering at Brunel University in 1990. He spent 5 years working in Systems Engineering at G.E.C Marconi Radar before joining I-Logix, Inc. as a Technical Consultant. Currently he is a on-site consultant at the Jet Propulsion Laboratory where he advises his customers in all aspects of improving their system and software engineering process and tool use.*